Minimal ‘main(void)’

```c
// This file (repl.c) is a part of Julia. License is MIT: http://julialang.org/license

// Standard headers
#include <string.h>
#include <stdint.h>
// Julia headers (for initialization and gc commands)
#include "uv.h"
#include "julia.h"

int main(int argc, char *argv[])
{
    // Initialize Julia
    uv_setup_args(argc, argv);
    libsupport_init();
    // Setup any runtime options here
    jl_options.fast_math = JL_OPTIONS_FAST_MATH_OFF; // --math-mode=ieee
    julia_init(JL_IMAGE_JULIA_HOME);
    // Run the REPL
    jl_function_t *start_client = (jl_function_t*)jl_get_global(jl_base_module, jl_symbol("_start"));
    (void)jl_apply(&start_client, 1);
    // Cleanup and graceful exit
    jl_atexit_hook(0);
    return 0;
}
```

This file is a part of Julia. License is MIT: http://julialang.org/license

More Standard Library

coreimg.jl

```
Main.Core.eval(Main.Core, :(baremodule Inference
using Core.Intrinsics
import Core: print, println, show, write, unsafe_write, STDOUT,
STDERR
ccall(:jl_set_istopmod, Void, (Bool,), false)

eval(x) = Core.eval(Inference,x)
eval(m,x) = Core.eval(m,x)

include = Core.include

## Load essential files and libraries
include("essentials.jl")
include("generator.jl")
include("reflection.jl")
include("options.jl")

# core operations & types
typealias Cint Int32
typealias Csize_t UInt
include("promotion.jl")
include("tuple.jl")
include("range.jl")
include("expr.jl")
include("error.jl")

# core numeric operations & types
include("bool.jl")
include("number.jl")
include("int.jl")
include("operators.jl")
include("pointer.jl")
const checked_add = +
const checked_sub = -
if !isdefined(Main, :Base)
    # conditional to allow redefining Core.Inference after base exists
```

sysimg.jl

```
baremodule Base
using Core.Intrinsics
ccall(:jl_set_istopmod, Void, (Bool,), true)
include = Core.include
include("coreio.jl")

eval(x) = Core.eval(Base,x)
eval(m,x) = Core.eval(m,x)

include("exports.jl")

## Load essential files and libraries
include("essentials.jl")
include("base.jl")
include("generator.jl")
include("reflection.jl")
include("options.jl")

# core operations & types
include("promotion.jl")
include("tuple.jl")
include("range.jl")
include("expr.jl")
include("error.jl")

# core numeric operations & types
include("bool.jl")
include("number.jl")
include("int.jl")
include("operators.jl")
include("pointer.jl")
include("refpointer.jl")
(::Type{T}){T}(arg) = convert(T, arg)::T
include("checked.jl")
importall .Checked
```